Vol. 4, No. 11, Jul-Sep 2024,(67-87)
Journal of New Achievements in Electrical, Computer and Technology (JNAECT)
مجله دستاوردهای نوین در برق،کامپیوتر و فناوری
دوره ۴، شماره ۱۱، تابستان ۱۴۰۳ (۸۷-۶۷)
journal homepage: https://jnaect.ir

یک الگوریتم جدید برنامه‌ریزی مبتنی بر ژنتیک در سیستم‌های پراکنده ناهمگن

[1]علیرضا عبدالحسینی (نویسنده مسئول مقاله)، [2]رضا روشنی(نویسنده دوم مقاله)
[1]موسسه آموزش عالی غیردولتی انتفاعی لامعی گرگانی،،گرگان،ایران Aabdolh@airport.ir
[2]موسسه آموزش عالی غیردولتی انتفاعی لامعی گرگانی،،گرگان،ایران
گروه مهندسی کامپیوتر،دانشگاه ملی مهارت،تهران، ایران
r.roshany@gmail.com

چکیده

سیستم های توزیع شده مانند Grid- و Cloud Computing خدمات وب را در سراسر جهان به کاربران خود ارائه می‌دهند. یکی از مهم ترین نگرانی هایی که ارائه دهندگان این خدمات با آن مواجه هستند، رسیدگی به هزینه کل مالکیت (TCO) است. بخش بزرگی از TCO مربوط به مصرف برق به دلیل مدیریت ناکارآمد منابع است. ماژول زمان‌بندی کار به عنوان یک جزء کلیدی می‌تواند تأثیر زیادی بر زمان پاسخ کاربر و استفاده از منابع اساسی داشته باشد. چنین سیستم های توزیع ناهمگنی، پردازنده های مختلف را با سرعت و معماری های متفاوت بکار برده اند. همچنین، برنامه کاربری که معمولاً به صورت گراف غیر چرخه ای جهت دار (DAG) ارائه می شود باید بر روی این نوع سیستم های پردازش موازی اجرا شود. از آنجایی که زمان‌بندی کار در چنین سیستم‌های پیچیده‌ای جزء مسائل NP-hard است رویکردهای اکتشافی موجود، دیگر کارآمد نیستند. بنابراین، روند کار استفاده از رویکردهای فراابتکاری ترکیبی است. در این مقاله، ما یک الگوریتم زمان‌بندی کار مبتنی بر ژنتیک به هم ریخته فراابتکاری را ارایه داده‌ایم تا زمان کل اجرا و طول زمان برنامه‌های کاربر را به حداقل برسانیم. در این راستا، ما از روش‌های اکتشافی دیگری مانند *سریعترین زمان پایان ناهمگن* (HEFT) برای تولید جمعیت اولیه هوشمند با استفاده از یک عملگر ترکیبی جدید بهره می‌بریم که برای کاوش افراد امکان‌پذیر و امیدوارکننده در فضای جستجو، ثروت زیادی ایجاد می‌کند. ما همچنین سایر اپراتورهای ژنتیکی را به روش صحیح هدایت می کنیم تا راه حل نهایی نزدیک به بهینه را تولید کنیم. برای رسیدن به نتایج ملموس ما چندین سناریو را انجام داده ایم. الگوریتم پیشنهادی ما در مقایسه با سایر رویکردهای موجود مانند نسخه‌های HEFT و QGARAR از نظر میانگین طول زمان، عملکرد بهتری داشته است.

کلمات کلیدی: زمانبندی وظایف، محاسبات ابری، گراف غیر چرخه ای جهت دار (DAG)

1. مقدمه

سیستم‌های توزیع‌شده شامل محاسبات و منابع ذخیره‌سازی نامحدودی هستند که با شبکه‌های پرسرعت به هم مرتبط هستند. [1-3] کاربران از سیستم‌های توزیع‌شده مانند Grid Computing معروف به E-Science و Cloud Computing معروف به E-Commerce به جای تهیه منابع ارایه شده بسیار زیاد برای پوشش آن تنها در ساعت اوج کار و تقاضا بسیار زیاد استفاده می‌کنند. رایانش ابری و شبکه ای خدمات کشسانی و اقتصادی را برای مشترکین خود بر اساس پرداخت به ازای استفاده به منظور صرفه جویی در مقیاس ارائه می کند. به عنوان مثال، چندین پروژه دانشگاهی که ماهیت محاسباتی فشرده ای دارند، حجم عظیمی از منابع پردازشی را نیاز دارند که تهیه چنین سیستم هایی برای دانشگاه مقرون به صرفه نیست. یا شرکتی را در نظر بگیرید که می‌خواهد از حجم عظیمی از داده‌های مربوط به چندین ماه، گزارش روزانه از بی‌شمار تراکنش پشتیبان تهیه کند. در این خط، سرویس آمازون S3 می‌تواند مشکل فوق‌الذکر را با هزینه‌های کم بر اساس فضای ذخیره‌سازی (دلار/ماه/ گیگابایت) فراهم کند. از سوی دیگر، زمان پاسخ یکی از مهم ترین پارامترهای کیفیت خدمات (QoS)[1] در چنین سیستم هایی است زیرا کاربران در نهایت ارائه دهندگان را با کیفیت خدمات دریافتی پایین رها می کنند.[6] در این راستا، عنصر کلیدی که در زمان پاسخگویی و کارایی سیستم توزیع شده تعیین‌کننده است، ماژول زمان‌بندی است. اساساً، رویکرد زمان‌بندی ناکارآمد منجر به تخریب در استفاده از منابع و افزایش زمان کل اجرا می‌شود. بنابراین طراحی و اجرای چارچوب برنامه ریزی هوشمند و کارآمد در چنین سیستم هایی اجتناب ناپذیر است. معمولاً برنامه های کاربردی حاوی وظایف فرعی مختلفی هستند که ممکن است بین آنها وابستگی داشته باشند. این نوع کاربرد در قالب گراف غیر چرخه ای جهت دار (DAG) ارائه می شود. از سوی دیگر، زیرساخت زیربنایی در سیستم‌های توزیع‌شده، مرکز داده[2] نامیده می‌شود که در آن از ماشین‌های فیزیکی مختلف و سرورهای متصل به شبکه‌های محلی پرسرعت (LAN) تشکیل شده است. از نظر جغرافیایی نیز مقیاس پذیر است. به عبارت دیگر، مراکز داده حتی می توانند از طریق شبکه های گسترده (WAN) متصل شوند، اگرچه به نظر می رسد یک موجودیت منحصر به فرد، به اصطلاح شفاف برای کاربر است. زمان‌بندی در چنین سیستم‌هایی به تخصیص منابع محاسباتی به وظایف کاربر که در برنامه DAG[3] قرار می‌گیرند اشاره دارد. هدف به حداقل رساندن زمان اجرای کل همه وظایف مشروط به حفظ محدودیت‌های وابستگی بین وظایف فرعی است. بنابراین، مسئله فوق از نظر محاسباتی به دسته NP-hard تعلق دارد.

با این وجود، در ادبیات، زمانی که کاربر درخواست می کند یک برنامه کاربردی انجام شود. برنامه های کاربردی نمودار وظیفه، که به شکل DAG نشان داده شده اند، معمولاً در سیستم های توزیع شده ناهمگن با الگوریتم های زمان‌بندی لیست مرتب می‌شوند.[11] زمان‌بندی‌های فهرست، فهرست مرتب‌شده‌ای از وظایف فرعی را بر اساس اولویت از پیش تعریف‌شده ارائه می‌کنند. در واقع هر زیرکار بر اساس وزنی که الگوریتم به هر گره اختصاص می دهد در لیست قرار می گیرد. یکی از معروف‌ترین الگوریتم‌های زمان‌بندی فهرست اکتشافی، رویکرد پایان زودهنگام ناهمگن (HEFT)[4] است.[11] به عنوان مثال Upward Rank، Downward Rank و Level Rank سه نسخه متفاوت از الگوریتم های مبتنی بر HEFT هستند. لیست زمانبندی ها در دو مرحله اجرا می شوند.

[1] Quality of Service
[2] Datacenter
[3] Directed acyclic graph
[4] Heterogeneous Earliest Finish Time

1. در مرحله اول همانطور که ذکر شد، هر زیرکار که به عنوان یک گره در DAG ارائه می شود، در یک لیست مرتب شده بر اساس اولویت از پیش تعیین شده قرار می گیرد. لیست مرتب شده باید تضمین کند که لیست مرتب سازی توپولوژیکی است. به عبارت دیگر، باید محدودیت های تقدم را برآورده کند.
2. در مرحله دوم، زمان‌بند باید بهترین پردازنده را برای ترسیم وظایف فرعی روی آن پیدا کند. این بدان معنی است که الگوریتم باید پردازنده ای را پیدا کند که خروجی وظایف فرعی را با زودترین زمان پایان (EFT[5]) در فاز دوم ارائه می‌دهد. در این خط، الگوریتم های تکاملی در ادبیات برای حل چنین مسائل ترکیبی اتخاذ شده است.

اما به دلیل پیچیدگی روزافزون و نیز مشکلاتی در چنین سیستم‌هایی، رویکردهای اکتشافی موجود دیگر کارآمد نیستند. بنابراین، روند استفاده از رویکردهای فراابتکاری ترکیبی ارایه شده است. این دلیلی برای گسترش یک الگوریتم جدید مبتنی بر ژنتیک به هم ریخته فراابتکاری برای رفع کاستی های اکتشافی فعلی است. ما از سایر رویکردهای اکتشافی در الگوریتم ژنتیک پیشنهادی جدید خود بهره می بریم. به عنوان مثال، برای تشکیل افراد برای ایجاد جمعیت اولیه، ما یک صف با اولویت چندگانه ایجاد می کنیم و لیست های مرتب شده ای را که توسط نسخه های HEFT تولید می شوند درج می کنیم. سپس، لیست‌ها را بر اساس الگوریتم جدید خود به هم می‌زنیم تا هم از اکتشاف و هم بهره‌برداری در فضای جستجو استفاده کنیم. ما همچنین سایر اپراتورهای GA را به روشی درست هدایت می کنیم تا راه حل های نزدیک به بهینه و کم سربار را به دست آوریم. نتیجه پیاده سازی نشان می دهد که این یک تکنیک امیدوارکننده است. سهم اصلی این مقاله به شرح زیر است:

1. گسترش یک الگوریتم ابتکاری مبتنی بر ژنتیک جدید با استفاده از صف چند اولویتی به این ترتیب، می‌توانیم با اعمال عملگرshuffle افراد هوشمند را در جمعیت اولیه تولید کنیم. هم از اکتشاف و هم بهره برداری در فضای جستجو سود می برد.
2. اعمال رویکرد HEFT برای یافتن پردازنده مناسب که حداقل EFT را تضمین می کند.

بقیه مقاله فعلی به شرح زیر سازماندهی شده است. کارهای مرتبط در بخش 2 مورد بحث قرار می گیرند. چارچوب سیستم شامل مدل های سیستم و کاربرد در بخش 3 قرار داده شده است. تعریف HEFT در بخش 4 آورده شده است. بخش 5 به رویکرد پیشنهادی ما اختصاص دارد. شبیه سازی و ارزیابی کار ما در بخش 6 آورده شده است. در نهایت، بخش 7 نتیجه‌گیری و جهت آینده را ارائه می دهد.

[5] Earliest Finish Time

2. آثار مرتبط

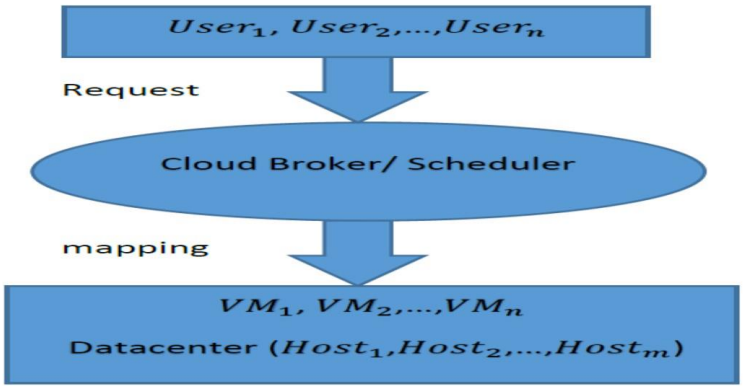
چندین کار در ادبیات برای ارائه راه حل روشن برای مشکلات زمان بندی کار در سیستم های توزیع شده انجام شده است. یک الگوریتم زمان‌بندی کار مبتنی بر PSO[6] برای مدیریت کارآمد منابع ابری در برنامه‌های علمی با توجه به مهلت تعیین‌شده توسط کاربر پیشنهاد شده است.[7] این یک تابع تناسب بر اساس کاهش زمان اجرا و هزینه تعریف شده بود.[7] اگرچه الگوریتم پیشنهادی امیدوارکننده است، اما به طور خاص برای مسائل علمی که فقط محاسبات فشرده هستند طراحی شده است. یکی دیگر از الگوریتم‌های زمان‌بندی کار مبتنی بر بهینه‌سازی کلنی مورچه‌ها (ACO[7]) برای به حداقل رساندن حجم وظایف *ارائه‌شده* در محیط ابری استفاده شد.[8] الگوریتم پیشنهادی کارایی خوبی در محاسبات ابری داشته است زیرا ماشین مجازی مناسب را بر اساس وظایف درخواستی راه اندازی می کند. با این وجود، الگوریتم فقط برای کارهای مستقل محدود است. یک الگوریتم زمانبندی کار مقرون به صرفه و کم هزینه بر اساس دو استراتژی عمل می کند. استراتژی اول بهترین ماشین های مجازی را به وظایف مبتنی بر تسلط پارتو اختصاص می دهد. استراتژی دوم با ترسیم وظایف بی اهمیت در فاز بعدی، هزینه کلی را کاهش می دهد.[9] تمرکز اصلی الگوریتم زمان‌بندی مقاله بر دیدگاه اقتصادی بود که ممکن است زمان پاسخگویی کاربر را قربانی کند. زمان‌بندی وظایف چند معیاره در سیستم‌های توزیع‌شده بر اساس فازی TOPSIS در 30امین کنفرانس IEEE کانادا در مهندسی برق و کامپیوتر (CCECE) ارائه شده است.[12] در این کار، نویسندگان از رویکرد فازی TOPSIS برای رتبه‌بندی گزینه‌ها بر اساس ترجیحات کاربران و ارائه‌دهندگان استفاده کردند. در اصل برای برنامه های بلادرنگ[8] که دارای محدودیت زمانی و مستقل هستند طراحی شده است. دیگر مبتنی بر اکتشاف به نام Bacteria Foraging برای زمان‌بندی وظایف مستقل در محیط محاسبات ابری پیشنهاد شده است.[14]

با وجود کارایی آن، برای برنامه‌هایی با وظایف وابسته قابل اجرا نیست. یک الگوریتم ژنتیک کوانتومی با اصلاح زاویه چرخش در ادبیات برای برنامه‌ریزی وظایف وابسته در سیستم‌های توزیع‌شده مانند مراکز داده ابری ارائه شده است.[15] الگوریتم پیشنهادی اساسا یک رویکرد خوب برای محاسبات کوانتومی است. همچنین به جمعیت تولید شده تصادفی برای فاز اولیه متکی است که منجر به دورهای تکرار بیشتر برای رسیدن به معیارهای رضایت می شود. بررسی ادبیات نشان می‌دهد که یک رویکرد فراابتکاری ترکیبی مطلوب است که در آن از سایر رویکردها برای رسیدن به نتایج خوب از نظر بهینه و بهره‌برداری از اکتشاف و بهره‌برداری در جستجو بهره می‌برد.

[6] Particle swarm optimization
[7] Ant colony optimization
[8] online

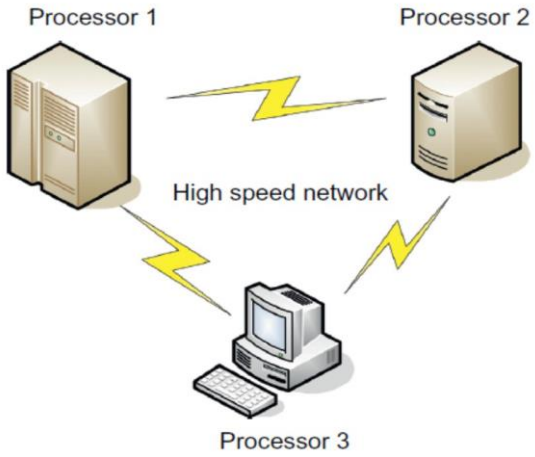
3. چارچوب سیستم

چارچوب سیستم پیشنهادی ما در شکل 1 نشان داده شده است. دارای اجزای مختلفی مانند ماژول Front-end، Cloud Broker، Scheduler، VMs و Datacenter است.



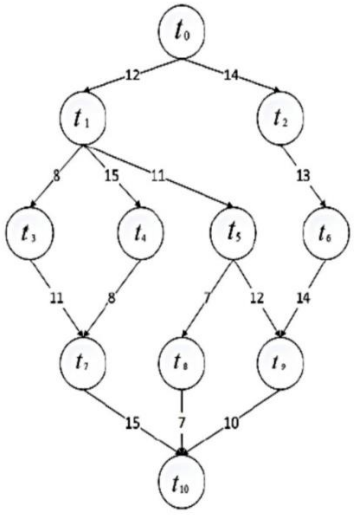
شکل 1. چارچوب سیستم پیشنهادی

ماژول Front-end برنامه های کاربردی کاربر را برای اجرا در ماشین های فیزیکی موازی دریافت می کند. Cloud Broker سرویس ابری و کیفیت خدمات مرتبط را به کاربران ارائه می دهد. مرکز داده شامل مجموعه ای از متر پردازنده های مختلف ناهمگن که با شبکه های با سرعت بالا متصل شده است. ناهمگونی بر اساس معماری و سرعت است. که هر کدام می توانند چندین VM را اجرا کنند. چنین مدل سیستمی در شکل 2 نشان داده شده است.



شکل 2. یک سیستم موازی متصل کامل با سه سیستم محاسباتی ناهمگن

علاوه بر این، مدل کاربردی در قالب DAG ارائه شده است. شکل 3 چنین برنامه هایی را نشان می دهد که وابستگی های متقابل وظایف فرعی دارند.



شکل 3. نمونه ای از برنامه DAG با 11 کار فرعی

هر نمودار وظیفه، یک DAG، دارای چندین گره است که برای محاسبات و لبه هایی تعریف شده اند که میانگین هزینه ارتباطی بین گره‌ها را نشان می دهد. لبه همچنین محدودیت تقدم بین گره ها را نشان می دهد. هر کار فرعی باید روی یک گره محاسباتی از سیستم موازی هدف ما اجرا شود. همچنین، هر DAG دارای دو گره خاص است که به ترتیب دارای گره های پیشین و جانشین نیستند. از آنجایی که سیستم تعریف شده ماهیت ناهمگن دارد، پردازش هر زیرکار DAG بر روی گره های محاسباتی مختلف هزینه متفاوتی دارد. به عنوان مثال، جدول 1 زمان های مختلف اجرای کار را در گره های پردازشی مختلف نشان می دهد. همچنین ستون آخر میانگین زمان اجرا را نشان می دهد.

Tasks	Processors			Average computation cost
	P_0	P_1	P_2	
t_0	7	9	8	8
t_1	10	9	14	11
t_2	5	7	6	6
t_3	6	8	7	7
t_4	10	8	6	8
t_5	11	13	15	13
t_6	12	15	18	15
t_7	10	13	7	10
t_8	8	9	10	9
t_9	15	11	13	13
t_{10}	8	9	10	9

جدول 1. زمان های مختلف اجرای کار در گره های پردازشی مختلف

4. زودترین زمان پایان ناهمگن (HEFT)

لیست زمانبندها الگوریتم های زمانبندی معروف در سیستم های توزیع شده هستند. زودترین زمان پایان ناهمگن (HEFT) به دسته زمانبندی لیست تعلق دارد. اولین بار توسط Topcuoglu و همکاران برای زمان‌بندی کار استاتیک در سیستم‌های پردازش موازی ناهمگن محدود[11] معرفی شد.از سوی دیگر، محاسبات شبکه ای در طبیعت ثابت است در حالی که رایانش ابری یک پارادایم پویا است. علاوه بر این، زمان‌بندی می‌تواند ثابت یا پویا باشد. زمان‌بندی استاتیک اطلاعات قبلی را اعمال می‌کند و با زمان تغییر نمی‌کند در حالی که یک زمان‌بندی پویا اطلاعات زمان اجرا وضعیت سیستم را اعمال می‌کند.[15] برای اعمال چنین الگوریتم ایستاگرا در محیط پویا، می‌توانیم پنجره زمانی استاتیک را برای درمان مشکل با مد استاتیک تعیین کنیم.[6] HEFT دو عملکرد مهم را اعمال می کند: زودترین زمان پایان (EFT) و زودترین زمان شروع (EST) اولی نشان‌دهنده اولین زمانی است که یک پردازنده P_j می‌تواند زیرکار T_i را اجرا کند، در حالی که دومی نشان‌دهنده اولین زمانی است که می‌توان اجرا را شروع کرد. اولین زمان شروع برای کار ورود در DAG صفر است که در آن معادله (1) محاسبه می شود. علاوه بر این، توابع EST و EFT برای سایر گره ها به ترتیب با معادلات (2) و (3) محاسبه می شوند.[11]

$$EST(n_{entery}, p_j) = 0 \quad (1)$$

$$EST(n_i, p_j) = \max\left\{avail\{j\}, \max\left(AFT(n_m) + c_{m,i}\right)_{n_m \in pred(n_i)}\right\} \quad (2)$$

$$EFT(n_i, p_j) = w_{i,j} + \mathrm{EST}\left(n_i, p_j\right) \quad (3)$$

تابع $pred(n_i)$ در معادله (2) مجموعه ای از تمام گره های پیشین در DAG را نشان می دهد. علاوه بر این، این اصطلاح $avail\{j\}$ زمانی را نشان می‌دهد که پردازنده p_j آخرین وظیفه را بر روی خود انجام داده و برای اجرای کار بعدی آماده است. *حداکثر* درونی در معادله (2) به این معنی است که زمان پایان واقعی (AFT) آخرین کار در $pred(n_i)$ باید تعیین شود. در همین حال، *حداکثر* بیرونی نشان می دهد که ممکن است در شرایطی اتفاق بیفتد که خروجی آخرین کار *فرعی* n_i در $pred(n_i)$ دیرتر از $avail\{j\}$ آن آماده شود. به عبارت دیگر، با وجود آمادگی p_j، زمان اجرا به زمانی موکول می شود که آخرین کار فرعی n_i آماده باشد، زیرا این رویه از تغییر در محدودیت های وابستگی در نمودار وظیفه DAG جلوگیری می کند. از طرف دیگر، اگر مقدار $avail\{j\}$ از زمان پایان آخرین کار *فرعی* در $pred(n_i)$ *بیشتر باشد*، با وجود اجرای همه زیرکارها در $pred(n_i)$ اجرای واقعی زمانی که پردازنده p_j آماده شود، آغاز می شود. پارامتر $c_{m,i}$ میانگین زمان انتقال بین پردازنده‌های p_m و p_i را نشان میدهد. اگر $m = i$ باشد آنگاه $c_{m,i} = 0$ است. زمان اجرای واقعی برای کار فرعی n_i بر روی پردازنده p_j با معادله (4) محاسبه می شود .[11] علاوه بر این، کل زمان اجرای DAG به نام *makepan* با معادله (5) محاسبه می شود.[11]

$$AFT\left(n_i, p_j\right) = min_{1 \le l \le m} EFT\left(n_i, p_l\right) \quad (4)$$

$$makespan = \max\left\{AFT\left(n_{exit}\right)\right\} \quad (5)$$

الگوریتم فوق در دو مرحله اجرا می شود. در مرحله اول وظایف فرعی بر اساس اولویت از پیش تعیین شده مرتب می‌شوند. رتبه‌بندی رو به بالا، رتبه‌بندی نزولی و رتبه‌بندی سطح سه رویکرد معمولی در این حوزه هستند.[10] به عنوان مثال، در رویکرد

رتبه‌بندی رو به بالا، به هر زیرکار وزنی از تکلیف فرعی *خروجی* تا زیرکار *ورودی* اختصاص داده می‌شود. این مقدار برای کار فرعی *خروج* با معادله (6) [11] محاسبه می شود. برای سایر وظایف فرعی، مقدار با معادله (7) به صورت بازگشتی محاسبه می‌شود[11] .

$$rank_u\left(n_{exit}\right)=\overline{w_{exit}} \tag{6}$$

$$rank_u\left(n_i\right)=\overline{w_i}+max_{n_j\in succ(n_i)}\left(\overline{c_{i,j}}+rank_u\left(n_j\right)\right) \tag{7}$$

علاوه بر این، میانگین زمان اجرای هر گره با رابطه (8) [11] محاسبه می شود. تابع *succ( )* به همه جانشینان در برنامه DAG نشان می دهد.

$$\overline{w_i}=\sum_{j=1}^{q} w_{i,j}\Big/ q \tag{8}$$

از طرف دیگر، در رویکرد نزولی، ارزش اولویت از گره *ورودی* به گره *خروجی* محاسبه می شود. این مقدار برای زیرکار *ورودی* صفر در نظر گرفته می‌شود در حالی که برای سایر وظایف فرعی مقدار به صورت بازگشتی با معادله (9) محاسبه می‌شود[11] .

$$rank_d\left(n_i\right)=max_{n_j\in pred(n_i)}\left(rank_d\left(n_j\right)+\overline{w_j}+\overline{c_{i,j}}\right) \tag{9}$$

در رویکرد رتبه بندی سطح، سطح برای هر کار فرعی با معادله (10) محاسبه می شود.[11] سپس، هر زیرکار بر اساس مقدار سطح آن به ترتیب افزایش در فهرست مرتب‌سازی قرار می‌گیرد. در صورت وجود سطح یکسان برای دو کار فرعی متفاوت، وظیفه فرعی که دارای ارزش بیشتری در مجموع مقادیر رتبه به بالا و پایین است در لیست مرتب شده از چپ به راست انتخاب می شود.

$$Level\left(T_i\right)=\begin{cases}0, & if\ T_i=T_{entery}\\ \max\left(Level\left(T_j\right)\right)_{T_j\in pred(T_i)}+1, & otherwise\end{cases} \tag{10}$$

به عنوان مثال، برای DAG نشان داده شده در شکل 3، جدول 2 مقادیر اولویت رویکردهای مختلف را برای هر گره نشان می دهد.

Tasks	$rank_u(t_i)$	$rank_d(t_i)$	Level	$rank_u(t_i) + rank_d(t_i)$
t_0	102	0	0	102
t_1	79	20	1	99
t_2	80	22	1	102
t_3	52	39	2	91
t_4	50	46	2	96
t_5	57	42	2	99
t_6	61	41	2	102
t_7	34	62	3	96
t_8	25	62	3	87
t_9	32	70	3	102
t_{10}	9	93	4	102

جدول 2. اولویت کار بر اساس سه رویکرد

از این رو، سه صف از وظایف فرعی $[t_0, t_2, t_1, t_6, t_5, t_3, t_4, t_7, t_9, t_8, t_{10}]$ ، $[t_0, t_1, t_2, t_3, t_6, t_5, t_4, t_7, t_9, t_8, t_{10}]$ و $[t_0, t_2, t_1, t_6, t_5, t_4, t_3, t_9, t_7, t_8, t_{10}]$ مرتب‌سازی توپولوژیکی معتبری هستند که بر اساس رویکردهای رتبه‌بندی رو به بالا، پایین و سطح [10] است. همانطور که قبلاً گفته شد، در مرحله دوم، یک کار فرعی از جلوی لیست مرتب شده برای برنامه ریزی انتخاب می شود. الگوریتم جستجو می کند تا پردازنده ای را بیابد که زودترین زمان پایان را برای آن کار فرعی تضمین کند. بنابراین، حالت های مختلف توسط DAG ورودی کوچک تولید می شود. حتی بدتر از آن، زمانی که DAG ورودی بزرگ است، نمی توان آن را با الگوریتم های قطعی مهار کرد. دلیل اینکه چرا ما رویکرد اکتشافی را برای کشف چنین مشکلی توسعه می دهیم.

5. الگوریتم ژنتیک پیشنهادی (GA[9]) برای زمان‌بندی وظایف در سیستم‌های ناهمگن

در این بخش، روش اکتشافی پیشنهادی خود را ارائه می‌کنیم که از سایر رویکردها، یعنی رتبه‌بندی رو به بالا، پایین و سطح در جمعیت اولیه خود بهره می‌برد. یکی از قدیمی ترین و کاربردی ترین روش های اکتشافی، الگوریتم ژنتیک (GA) است که از طبیعت الهام می گیرد. به عبارت دیگر، هر فردی که خود را با شرایط محیطی وفق دهد زنده خواهد ماند. بنابراین، مشکلات شبیه سازی شده می توانند بر این اساس کار کنند.

هر راه حل کاندید، فردی درGA ، که مجاورت بیشتری با راه حل بهینه دارد، در نسل بعدی باقی خواهد ماند. GA چند عملگر برای تولید تولید مناسب دارد. روش پیشنهادی ما در شکل 4 نشان داده شده است.

Input: GA and DAG characteristics
Output: An optimal task scheduling
1. Call **INIT** procedure to create initial population
2. **Repeat**
3. Call **Mapper** procedure to apply task-to-processor mapping and evaluate fitness
4. Copy the elitism individuals directly to the next generation
5. **Repeat**
6. Call **Roulette-wheel** operator to select candidates
7. Call **Crossover** operator
8. Call **Mutation** operator
9. **Until** the new population is completed
10. Replace the old population with new one
11. **Until** the termination criteria are met
12. **Return** an optimal schedule

End

شکل 4. الگوریتم روش پیشنهادی

5.1 ژن ها، کروموزوم ها و روش INIT

در GA، فنوتیپ باید به ژنوتیپ تبدیل شود. در این راستا، رمزگذاری دلخواه را می توان گسترش داد. در این مقاله، هر یک از وظایف فرعی را می توان به عنوان یک ژن استفاده کرد. بنابراین، دنباله ای از ژن ها یک کروموزوم می سازد. به عنوان مثال، [ ] را می توان به عنوان یک کروموزوم معتبر شناخته شد زیرا هر زیرکار پس از والدینش بازدید می شود. برای ایجاد جمعیت اولیه، ما رویه INIT را برای ایجاد نسل فعلی با افراد ثابت *Popsize* فراخوانی میکنیم. در این کار ما *Popsize* =100 را می‌گیریم. برای بهره مندی از روش های دیگر، ما سه کروموزوم از رتبه بندیUpward، Downward و Level می گیریم. بقیه کروموزوم ها به طور تصادفی ایجاد شده و از چند صف با توزیع مناسب به هم ریخته می شوند زیرا پراکندگی ناکارآمد افراد بر بهینه بودن، سرعت و همگرایی الگوریتم تأثیر زیادی دارد. کد شبه روش INIT در شکل 5 نشان داده شده است. همچنین از یک روش Shuffle استفاده می کند که در شکل 6 به تفصیل شرح داده شده است. همچنین در شکل 6، رویه ای که در خط 18 قابل انجام است بررسی می کند که آیا کروموزوم معتبر است یا خیر.

[9] Genetic Algorithm

به طور خلاصه، این روش بررسی می‌کند که کروموزوم، فهرست ژن‌ها، مرتب‌سازی توپولوژیکی است یا نه.

Procedure **INIT**
Input: A DAG with its characteristics, *PopSize* and *n* are as length of Population and chromosome respectively
Output: An initialize Population with *PopSize*
1. Size=1;
2. Ind1= Call HEFT-UpwardRank to make first individual
3. Add Ind1 into Population
4. Size=size+1;
5. Ind2= Call HEFT-DownwardRank to make second individual
6. Add Ind2 into Population
7. Size=size+1;
8. Ind3= Call HEFT-LevelRank to make third individual
9. Add Ind3 into Population
10. Size=size+1;
11. Pop1= **Shuffle** (Population(1) , n)
12. Pop2= **Shuffle** (Population(2) , n)
13. Pop3= **Shuffle** (Population(3) , n)
14. Add *PopSize*/5 of individuals from Pop1 to Population so that containing its first individual.
15. Add *PopSize* /5 of individuals from Pop2 to Population so that containing its first individual.
16. Add *PopSize* /5 of individuals from Pop3 to Population so that containing its first individual.
17. For the rest, generate random individuals to fill Population with *PopSize* size.

Return *Population* with *PopSize*

شکل 5. روش INIT برای ایجاد جمعیت اولیه

Procedure **Shuffle (*Individual, n*)**
Input: An *individual* and *n* as length of its chromosome size
Output: A list containing *k* members of individuals known as *CurrentPop* with *CurrentPopSize.*
1. *CurrentPopSize*=1;
2. *CurrentPop*= [];
3. Add *Individual* to *CurrentPop*
4. Copy *Individual to* T
5. For i=1 to n-1 do
6. Find - as the first successor of T_i
7. Temp = T_i
8. L=k-1;
9. For q=i to k-2 do
10. $T_q = T_{q+1}$
11. End-For
12. T_L = *Temp*
13. If **feasible**(new individual) then
14. Add new individual to CurrentPop
15. *CurrentPopSize* = *CurrentPopSize*+1;
16. End-if
17. // Again it repeats for original individual to make new one
18. Copy *Individual to* T
19. Find T_j as the last predecessor of T_i
20. L=j+1;
21. Temp = T_i.
22. For q=i down to j+2 do
23. $T_q = T_{q-1}$
24. End-For
25. T_L = Temp
26. If **feasible**(new individual) then
27. Add new individual to *CurrentPop*
28. *CurrentPopSize*=*CurrentPopSize*+1;
29. End-if
30. End-For

Return *CurrentPop* with *CurrentPopSize*

شکل 6. روش Shuffle برای کشف فضای جستجو

5.2 رویه نقشه برداری

در این مرحله، هر کروموزوم که نماینده یک راه حل کاندید است را می توان با الگوریتم HEFT ، در 2. 2، روی پردازنده های چندگانه نگاشت کرد. سپس، *makepan* می تواند به عنوان یک ارزش تناسب برای ارزیابی در نظر گرفته شود. یعنی، هر کروموزوم با کمترین گستردگی، مناسب‌ترین کروموزوم است.

5.3 رویه رولت–چرخ

بخشی از افراد با احتمالات خاص برای ایجاد نسل جدید انتخاب می شوند. این توسط اپراتور متقاطع اتفاق می افتد. در چرخ رولت، روشی را در نظر می گیریم که در آن هر کروموزوم با تناسب بالا شانس بیشتری برای انتخاب دارد. معادله (11)، فرمولی است که شانس انتخاب هر کروموزوم را نشان می دهد.

$$p_i = \frac{fitness_i}{\sum_{j=1}^{PopSize} fitness_j} \qquad (11)$$

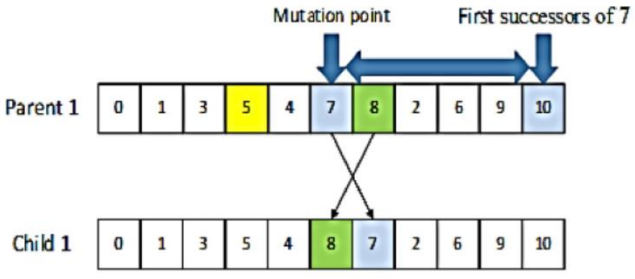
5.4 اپراتور متقاطع

برای ایجاد نسل جدید، GA از عملگر متقاطع استفاده می کند. پس از انتخاب چند نفر توسط اپراتور چرخ رولت، افراد برای اپراتور متقاطع وارد می شوند. از ابتدا، یک عدد تصادفی در [1. . chromosome-length] تولید می شود. سپس والدین را در دو بخش به نصف تقسیم می کند. قسمت چپ والد اول برای قسمت چپ فرزند اول قرار می گیرد. سپس قسمت راست کودک توسط تیره‌هایی پر می‌شود که در هنگام بازدید از تیره والدین دوم از چپ به راست در این کودک ظاهر نمی‌شود. بر این اساس، قسمت چپ والد دوم به سمت چپ فرزند دوم قرار می گیرد. بقیه با تیره‌هایی پر می‌شود که در این کودک وقتی که تیره والدین اول از چپ به راست بازدید می‌شود، ظاهر نمی‌شوند. در شکل 7 به خوبی نشان داده شده است. در این مرحله، نرخ متقاطع، معروف به جعبه ، 80 درصد *PopSize* را در نظر می گیریم.

شکل 7. عملگر متقاطع در GA

5.5 عملگر جهش

اکثر الگوریتم های تکاملی در بهینه محلی گیر می کنند. GA از این رویداد مستثنی نیست. برای جلوگیری از این پدیده، GA از عملگر جهش بهره می برد تا فرصتی برای باز کردن اتاق جدید برای جستجوی فضای جستجوی ناشناخته ایجاد کند که ممکن است راه حل خوبی داشته باشد. در جهش، یک ژن با عنوان t_i به‌طور تصادفی انتخاب می‌شود، سپس اولین جانشین این ژن، وظیفه فرعی، با عنوان t_j می‌باشد. سپس ژن تصادفی، زیروظیفه t_k، که در آن k متعلق به [i+1. . j-1] است، می‌توان با توجه به این که تمام کارهای فرعی پیش از این در فهرست مرتب‌سازی شده جلوتر هستند، انتخاب شود. سپس، وظایف فرعی t_i و t_k و می تواند جایگزین شوند. به این ترتیب یک عضو جدید تولید می شود. شکل 8 برای نشان دادن عملگر جهش در GA نشان داده شده است. همانطور که شکل 8 نشان می دهد، زیرکار t_7 در موقعیت ششم است به طور تصادفی انتخاب شده است. اولین جانشین در DAG است. سپس، ژن تصادفی، وظیفه فرعی t_k، باید در موقعیت $k \in [7. . 10]$ انتخاب شود تا همه نسل های قبلی t_k در کروموزوم جلوتر از t_7 باشند. به عنوان مثال، در موقعیت هفتم وظیفه فرعی t_8 انتخاب می‌شود، زیرا تنها سلف آن، یعنی t_5 در کروموزوم جلوتر از t_7 است. در نتیجه، ژنهای t_7 و t_8 می توانند جایگزین شود. در این مرحله، نرخ جهش، معروف به $mrate$، 20 درصد از $PopSize$ را در نظر می گیریم.



شکل 8. عملگر جهش در GA

5.6 ضوابط خاتمه

GA یک الگوریتم بی پایان است مگر اینکه کاربر برنامه را قطع کند که به نقطه خاصی می رسد. معیارهای خاتمه معمول در چنین الگوریتم هایی دورهای از پیش تعیین شده، رسیدن به تابع تناسب مناسب و غیره است. در این الگوریتم ما برنامه را 100 بار تکرار می کنیم. عدد 100 بصورت تجربی بدست می آید.

6. شبیه سازی و ارزیابی

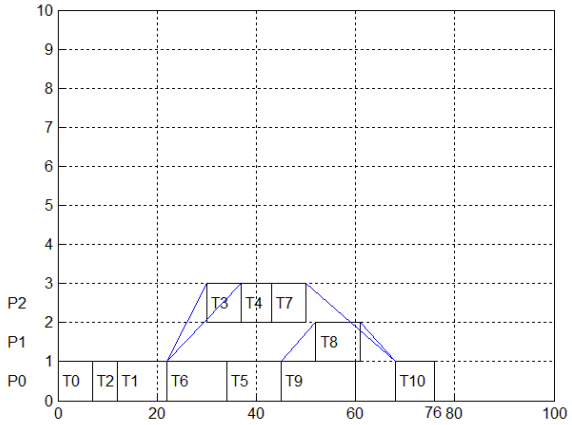
در این بخش، مجموعه‌ای از آزمایش‌ها را برای ارزیابی اثربخشی روش پیشنهادی خود اعمال کردیم. در این راستا، ما کار خود را ارزیابی در مدت زمان اجرای کل، *makespan*، مربوط به گراف وظایف در برابر نسخه های عمده و رویکرد موجود مانند الگوریتم ژنتیک کوانتومی با پالایش زاویه چرخش برای برنامه ریزی کار وابسته به سیستم های توزیع شده شناخته شده به عنوان[15] QGARAR

6.1 محیط شبیه سازی

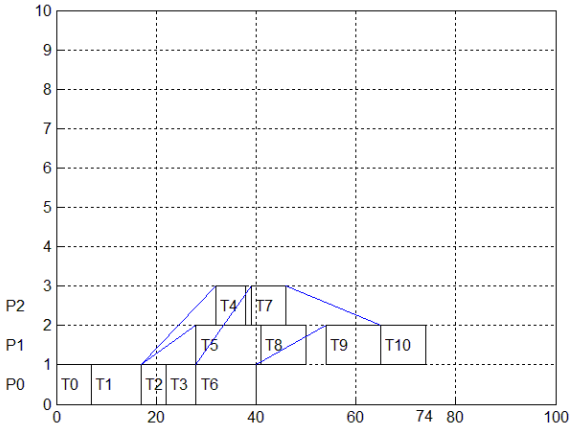
تمامی آزمایش ها با استفاده از لپ تاپ Sony VAIO با پردازنده 2. 26 گیگاهرتزی Intel Core 2 Duo و 4 گیگابایت رم و با استفاده از نرم افزار Matlab 2015 اجرا شد.

6.2 ارزیابی روش شناسی

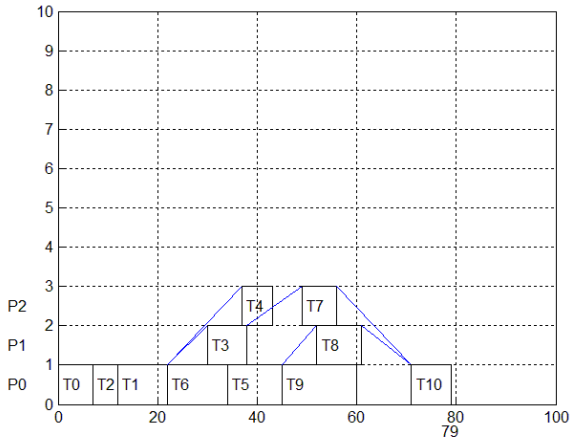
برای ارزیابی الگوریتم پیشنهادی، این بخش را در دو قسمت انجام می دهیم. ابتدا مثالی برای نشان دادن اثربخشی و کاربرد الگوریتم پیشنهادی می آوریم. ثانیا، سناریوهای پیچیده ای را برای رسیدن به نتایج ملموس تعریف می کنیم. در ابتدا، یک DAG نشان داده شده در شکل 3. ما برنامه خود را اجرا می کنیم و نتیجه را با سه الگوریتم HEFT اکتشافی مقایسه می کنیم. یعنی رویکردهای رتبه‌بندی رو به بالا، پایین، سطح [10] و یک جدیدترین اکتشافی مانند .[15] QGARAR شکل 9، شکل 10، شکل 11، شکل 12 و شکل 13 به ترتیب رده‌بندی به سمت بالا، پایین، سطح و QGARAR و رویکرد پیشنهادی ما را نشان می‌دهند. علاوه بر این، در این شکل ها، خطوط اریب نشان دهنده انتقال داده بین پردازنده های مختلف است.



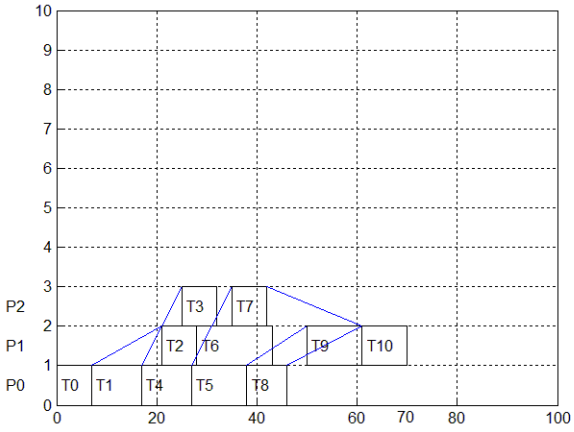
شکل 9. اولویت رتبه بندی صعودی [10]



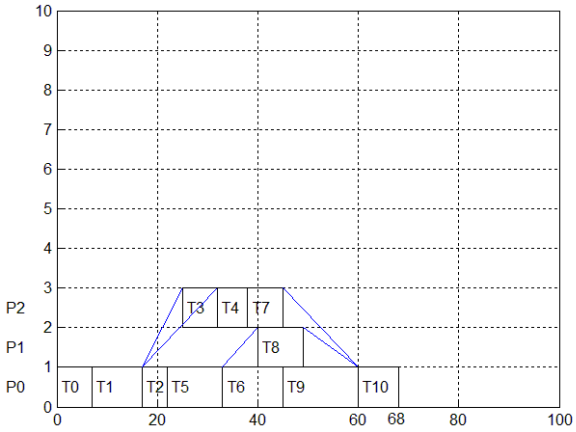
شکل 10. اولویت رتبه بندی نزولی[10]



شکل 11. اولویت رتبه بندی سطح [10]

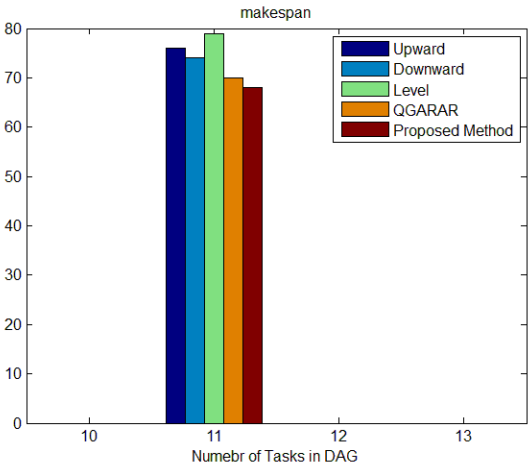


شکل 12. QGARAR [15]



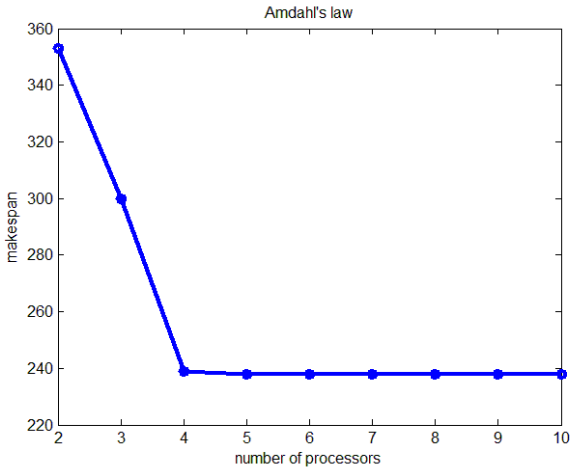
شکل 13. زمانبندی پیشنهادی ما

همچنین، شکل 14 نشان می دهد مقایسه کل زمان اجرا، *makespan*، بین روش برای DAG در شکل 3. به تصویر کشیده آن را به عنوان به تصویر می کشد، روش پیشنهادی ما کمترین *makespen* در مقایسه میان دیگران است.



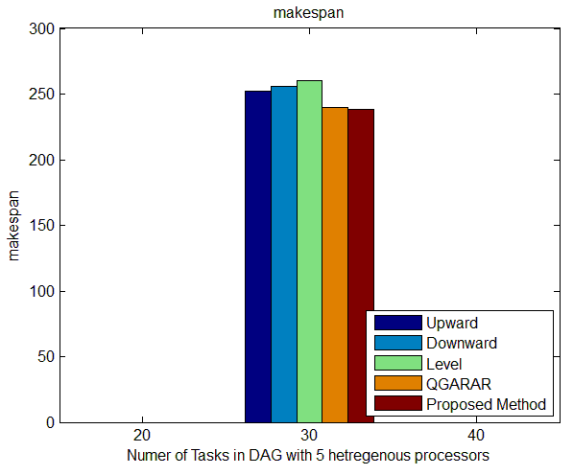
شکل 14. مقایسه فاصله بین رویکردهای DAG که در شکل 3 نشان داده شده است

ثانیا، برای رسیدن به نتایج ملموس، ما 9 سناریو مختلف را انجام می دهیم. ما 3 نوع نمودار را تعریف می کنیم. یعنی نمودارهای سبک، متوسط و سنگین با 30، 50 و 100 گره پردازشی که هر کدام به ترتیب بر روی 5، 8 و 10 پردازنده موازی ناهمگن اجرا می شوند. ما اعداد وزنی را که با توزیع یکنواخت در [10...20] برای گره های پردازش و اعداد وزن توزیع یکنواخت از [10...30] برای هزینه های ارتباطی بین پردازنده ها تولید می شوند، می گیریم. ظاهراً 9 سناریو تعریف می کنیم، اما به دلایلی که گفته می شود، تنها 3 سناریو پیچیده یعنی نمودارهای سبک، متوسط و سنگین را به ترتیب در 5، 8 و 10 پردازنده تحلیل می کنیم. به عنوان مثال، یک DAG نور تصادفی با 30 گره را انتخاب کنید که دارای ویژگی های محاسباتی و ارتباطی ذکر شده است. ما آن را روی پردازنده های موازی از 2 تا 10 پردازنده با افزایش یک پردازنده در هر مرحله اجرا می‌کنیم. ما ثبت شده اند *makespan* برای جلو-گفت DAG به عنوان شکل 15 نشان می دهد.



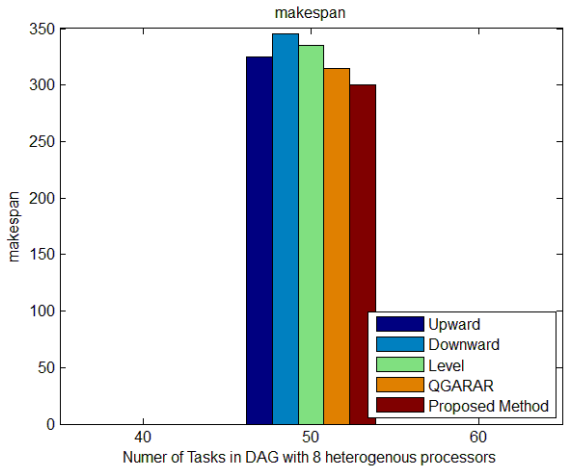
شکل 15. مسیر را با افزایش پردازنده ها می‌سازد

همانطور که در شکل نشان داده شده است به ترتیب 353، 300، 239 و 238 برای پردازنده های 2، 3، 4 و 5 می باشد. این نشان می‌دهد که اگر پردازنده‌های بیشتری را خرج کنیم، هیچ بهبودی در ساخت و ساز وجود ندارد. این یک دلیل واضح بر قانون امدال است زیرا وابستگی بین وظایف در برابر افزایش سرعت سرکوب می شود.[16] به عبارت دیگر، برای اعداد پردازنده 5 < تغییری در *makepan* وجود ندارد. به هر حال 3 سناریوی پیچیده 50 بار اجرا شده است. پس از آن، به طور متوسط *makespan* عنوان یک نتیجه از هر روش ثبت شد. شکل 16 تجزیه و تحلیل سناریوی اول را باDAG های تصادفی سبک در حال اجرا بر روی 5 پردازنده موازی ناهمگن نشان می دهد.



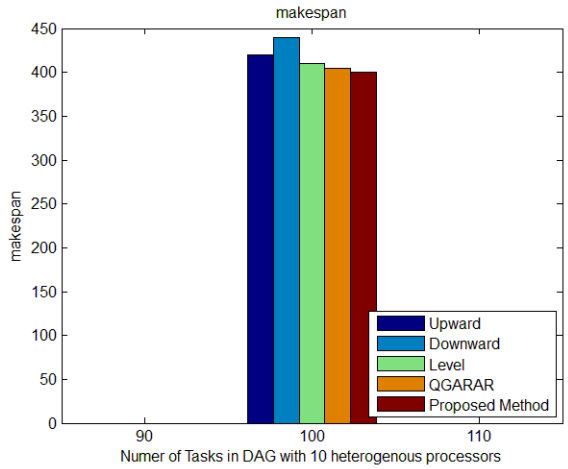
شکل 16. مقایسه رویکردها بر حسب طول ساخت در سناریوی اول

همچنین، شکل 17 تجزیه و تحلیل سناریوی دوم را باDAG های تصادفی متوسط در حال اجرا بر روی 8 پردازنده موازی ناهمگن نشان می دهد.



شکل 17. مقایسه رویکردها بر حسب طول ساخت در سناریوی دوم

در نهایت، شکل 18 تجزیه و تحلیل سناریوی سوم را باDAG های تصادفی سنگین در حال اجرا بر روی 10 پردازنده موازی ناهمگن نشان می دهد.



شکل 18. مقایسه رویکردها بر حسب طول ساخت در سناریوی سوم

از آنجایی که الگوریتم‌های اکتشافی و تکاملی قطعی نیستند، اما ماهیت تصادفی دارند، ما فقط نتیجه را با چندین اجرای شبیه‌سازی‌های مختلف تحلیل می‌کنیم. بنابراین، ما سناریوهای مختلفی را برای شبیه سازی انجام داده ایم. نتایج شبیه‌سازی برتری بالای رویکرد پیشنهادی در برابر نسخه‌های چندگانه HEFT و برتری حاشیه‌ای بر QGARAR را نشان می‌دهد. به دلیل استراتژی هایی است که الگوریتم پیشنهادی اتخاذ کرده است. در مرحله اولیه، به طور هوشمندانه جمعیت اولیه را از اکتشافی های مختلف برای بهره برداری از راه حل های کارآمد تشکیل داد و در مرحله بعدی فضای جستجو را برای کشف راه حل های امیدوارکننده تغییر داد. اگرچه QGARAR درمان خوبی دارد، رویکرد پیشنهادی ما نیمه تصادفی است و از مزایای اکتشاف و بهره‌برداری در فضای جستجو استفاده می‌کند. به همین دلیل است که الگوریتم پیشنهادی در همه سناریوها بر QGARAR برتری دارد.

7. نتیجه گیری و کار آینده

ماژول زمانبندی وظیفه یکی از مهم ترین اجزای سیستم های توزیع شده مانند رایانش شبکه و ابر است. تخصیص ناکارآمد منابع به دلیل زمان‌بندی ناکارآمد وظایف، مفهوم اقتصادی را هدایت نمی‌کند. زیرا منبع استفاده شده اساسی برای وظایف در حال اجرا منطقی نیست. از سوی دیگر، کاربران قطعاً به لطف بازار آزاد رقابتی مانند محیط چند ابری، از شر ارائه دهندگان ناکارآمد خلاص خواهند شد. برای بقا در این بازار رقابتی، ارائه دهندگان را وادار می کند تا زمان پاسخگویی را به عنوان یک عنصر کلیدی در پارامترهای کیفیت خدمات بهبود بخشند.

پاسخ‌دهی خوب از طریق زمان‌بندی هوشمند به دست می‌آید چرا که ما یک الگوریتم زمان‌بندی کار مبتنی بر ژنتیک را در چنین سیستم‌های ناهمگنی گسترش داده‌ایم. برای داشتن بهینه، سرعت و همگرایی خوب، از روش‌های اکتشافی دیگری بهره گرفته‌ایم که در آن به‌طور هوشمندانه چند صف را در جمعیت اولیه قرار می‌دهیم.

در نتیجه، رویکرد ما در مقایسه با سایر رویکردهای موجود، از نظر میانگین *ماندگاری* نتیجه بهتری داشته است. این به دلیل استراتژی هایی است که الگوریتم پیشنهادی اتخاذ کرده است. در مرحله اولیه، به طور هوشمندانه جمعیت اولیه را از اکتشافی های مختلف برای بهره برداری از راه حل های کارآمد تشکیل داد و در مرحله بعدی فضای جستجو را برای کشف راه حل های امیدوارکننده تغییر داد. اگرچه QGARAR درمان خوبی دارد، رویکرد پیشنهادی ما نیمه تصادفی است و از مزایای اکتشاف و بهره‌برداری در فضای جستجو استفاده می‌کند. به همین دلیل است که الگوریتم پیشنهادی در تمامی سناریوها بر QGARAR برتری دارد. ما در نظر داریم یک مدل زمان‌بندی هوشمند را گسترش دهیم که به طور همزمان زمان پاسخ و استفاده از منابع را با استفاده از مقیاس‌بندی فرکانس ولتاژ دینامیکی (DVFS[10]) برای کارهایی که هیچ محدودیت زمانی ندارند، بهبود می‌بخشد. به عبارت دیگر، ما قصد داریم الگوریتم زمانبندی را برای یافتن زمان سستی هر کار توسعه دهیم. با دانستن اطلاعات زمان هر کار می توانیم فرکانس و جفت ولتاژ را بر اساس حجم کاری فعلی تنظیم کنیم.

[10] Dynamic voltage frequency scaling

منابع:

[1] Armbrust M., Fox A., Griffith R., Joseph A. D. and Katz R., ‘‘Above the Clouds: A Berkeley View of Cloud Computing’’. Technical report EECS-2009-28, UC Berkeley, 2009.
[2] Mell P., Grance T., The NIST definition of cloud computing, Natl. Inst. Stand. Technol. 53 (6) (2009) 50.
[3] Tanenbaum A.S. And Steen M.V., distributed systems: principles and paradigms, second edition, prentice hall (2007).
[4] Hosseini Shirvani Mirsaeid, Rahmani A.M., Sahafi A. An iterative mathematical decision model for cloud migration: A cost and security risk approach. *Softw Pract Exper* . 2017; 1-37. https://doi.org/10.1002/spe.2528.
[5] Amazon FAQ on S3. https://aws.amazon.com/s3/faqs/ [6 Jaunarary 2018].
[6] Burkimsher A., Bate I., Indrusiak L. S., A survey of scheduling metrics and an improved ordering policy for list schedulers operating on workloads with dependencies and a wide variation in execution times, Future Generation Computer Systems 29 (2013) 2009–2025.
[7] Somasundaram T. S. and Govindarajan K., "CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud," Future Generation Computer Systems, vol. 34, pp. 47-65, 5// 2014.
[8] Wang L. and Ai L., "Task Scheduling Policy Based on Ant Colony Optimization in Cloud Computing Environment," in LISS 2012: Proceedings of 2nd International Conference on Logistics, Informatics and Service Science, Z. Zhang, R. Zhang, and J. Zhang, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 953-957.
[9] Su S., Li J., Huang Q., Huang X., Shuang K., and Wang J., "Cost-efficient task scheduling for executing large programs in the cloud," Parallel Computing, vol. 39, pp. 177-188, 4// 2013.
[10] Xu Y., Li K., Hu J., and Li K., "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," Information Sciences, vol. 270, pp. 255-287, 6/20/ 2014.
[11] Topcuoglu H., Hariri S., and Min-You W., "Performance-effective and low-complexity task scheduling for heterogeneous computing," Parallel and Distributed Systems, IEEE Transactions on, vol. 13, pp. 260-274, 2002.
[12] Hosseini Shirvani, M. , Amirsoleimani, N., Salimpour, S., Azab, A., Multi-Criteria Task Scheduling in Distributed Systems based on Fuzzy TOPSIS, 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE).
[13] Ghaffari A. and Kamalinia A., “Hybrid Task Scheduling Method for Cloud Computing by Genetic and PSO algorithm”, Journal of Information Systems and Telecommunication, 4(4), 2016.
[14] Verma J., Sobhanayak S., Sharma S., Turuk A. K. and Sahoo B., "Bacteria foraging based task scheduling algorithm in cloud computing environment," 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, 2017, pp. 777-782.
[15] Gandhi T., Nitin and Alam T., "Quantum genetic algorithm with rotation angle refinement for dependent task scheduling on distributed systems," 2017 Tenth International Conference on Contemporary Computing (IC3), Noida, India, 2017, pp. 1-5.
[16] Amdahl, Gene M., "Amdahl's Law in the Multicore Era", Computer, 41 (7), 33–38, 2008.
[17] Hosseini Shirvani, Mirsaeid, Web Service Composition in multi-cloud environment: A biobjective genetic optimization algorithm, IEEE international conference INISTA 2018, Thessaloniki, Greece.

A New Shuffled Genetic-based Task Scheduling Algorithm in Heterogeneous Distributed Systems

[1.]Alireza Abdul Hosseini (responsible author of the article). [2.]Reza Roshni (second author of the article)

[1.] lamei Gorgani Institue of Higher Education Gorgan,Iran
Aabdolh@airport.ir
[2.]lamei Gorgani Institue of Higher Education Gorgan,Iran
Department of Mechanical Engineering, National University of, Skill (NUS) ,Tehran, Iran
r.roshany@gmail.com

Abstract:

Distributed systems such as Grid- and Cloud Computing provide web services to their users worldwide. One of the most important concerns that service providers face is total cost of ownership (TCO). A large portion of TCO is related to power consumption due to inefficient resource management. Task scheduling module as a key component can have a great impact on user response time and underlying resource utilization. Such heterogeneous distributed systems have used different processors with different speeds and architectures. Also, the user program, which is usually represented as a directed acyclic graph (DAG), must be executed on these types of parallel processing systems. Since work scheduling in such complex systems is part of NP-hard problems, the existing heuristic approaches are no longer efficient. Therefore, the workflow is to use hybrid meta-heuristic approaches. In this paper, we have presented a meta-heuristic genetic shuffled task scheduling algorithm to minimize the total execution time and duration of user programs. In this regard, we take advantage of other heuristic methods such as Heterogeneous Fastest Termination Time (HEFT) to generate an intelligent initial population using a new hybrid operator, which creates a wealth for exploring feasible and promising individuals in the search space. We also direct other genetic operators in the correct way to produce a near-optimal final solution. To achieve tangible results, we have performed several scenarios. Compared to other existing approaches such as HEFT and QGARAR versions, our proposed algorithm has performed better in terms of average duration.

Keywords: Task Scheduling, Cloud Computing, Directed Acyclic Graph (DAG)